Bonjour Overview

Developer

Contents

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

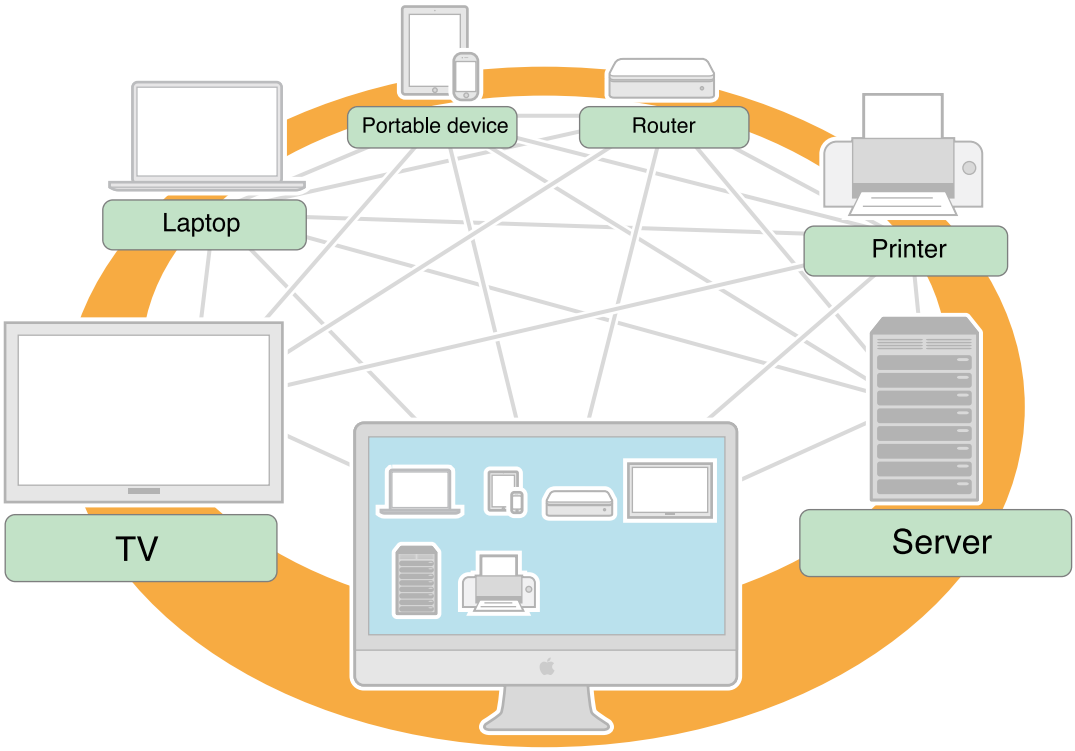
Figures and Tables

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

About Bonjour

The Bonjour zero-configuration networking architecture provides support for publishing and discovering TCP/IP-based services on a local area or wide area network. This document describes the Bonjour architecture at a high level and briefly describes what Bonjour APIs are available.

Important: This document is not meant as a programming guide. It is intended to provide a high-level overview. For more detailed coverage at the API level (including code snippets), you should read *DNS Service Discovery Programming Guide* or *NSNetServices and CFNetServices Programming Guide*.



At a Glance

Bonjour is Apple's implementation of a suite of zero-configuration networking protocols. Bonjour is designed to make network configuration easier for users.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

For example, Bonjour lets you connect a printer to your network without the need to assign it a specific IP address or manually enter that address into each computer. With zero-configuration networking, nearby computers can discover its existence and automatically determine the printer's IP address. And if that address is a dynamically assigned address that changes, they can automatically discover the new address in the future.

Apps can also leverage Bonjour to automatically detect other instances of the app (or other services) on the network. For example, two users running an iOS photo sharing app could share photos over a Bluetooth personal area network without the need to manually configure IP addresses on either device.

Bonjour Provides Efficient Service Discovery

The Bonjour protocol supports advertising and discovering services in a manner that is efficient and robust using multicast DNS (mDNS) and, when needed, link-local addressing.

Relevant Chapters: "Bonjour Concepts" (page 8)

Bonjour Reserves the .local Domain for mDNS-Advertised Services

Bonjour host names and service names are constructed using a specific set of rules.

Relevant Chapters: "Domain Naming Conventions" (page 17)

Bonjour Uses SRV, TXT, and PTR Records to Look Up Services

Bonjour uses service-specific records to advertise the existence of services. PTR records let you discover all of the services in a domain;SRV records translate a service instance name, type, and domain into a hostname and port; A and AAAA records translate a host name into an IP address, and TXT records provide additional information about a service.

Relevant Chapters: "Bonjour Operations" (page 23)

Bonjour Provides APIs at Multiple Layers in OS X and iOS

In OS X and iOS, Bonjour provides the ability to advertise and discover services using Foundation, Core Foundation, and C APIs. In OS X, Bonjour also provides a Java API. On other platforms such as Windows and Linux, Bonjour provides a C API.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Relevant Chapters: "Bonjour API Architecture" (page 21)

Prerequisites

This document assumes that you are already familiar with the networking concepts described in *Networking Overview* and *Networking Concepts*.

See Also

- *DNS Service Discovery Programming Guide* describes the Bonjour API appropriate for Darwin and Windows programmers and developers.
- *NSNetServices and CFNetServices Programming Guide* describes the Bonjour API appropriate for Cocoa programmers and C and C++ programmers on OS X and iOS.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Bonjour Concepts

Bonjour is a suite of protocols for zero-configuration networking over IP that Apple has submitted to the IETF as part of the ongoing standards-creation process. This section describes the problems that Bonjour solves and how it solves them.

Why Bonjour?

Over the past two decades, computers have gradually transitioned away from platform-specific protocols such as AppleTalk, IPX, and NetBIOS towards the Internet Protocol (IP). The majority of computers and other network devices all use TCP/IP for communication. In that transition, however, one piece of functionality was lost—the ability to add devices to a local network and then connect to those devices from computers and other devices on the network, all with little or no configuration.

For IP to work, each device needs a unique address, whether statically assigned or dynamically assigned by a DHCP server. A dynamically assigned address can change, so without Bonjour, printers and other devices had to be manually configured with a static address so that computers on the network could reach them. Then, a network administrator had to configure a DNS server so that computer users didn't have to connect to the printer by IP address. Thus, a seemingly minor task required significant configuration. Because people who do not fit the traditional role of the network administrator often set up networks—families connecting their laptops to the Internet over a shared router, for example—that level of configuration just isn't practical.

And even within managed networks run by IT professionals, it makes no sense to require manual configuration for devices like printers. People expect to be able to plug in the printer, plug two laptops together, or look for a file server or game server on the local network without wasting time trying to get the configuration right.

To support this, people need a simple and reliable way to configure and browse for services over IP networks. They want to discover available services and choose one from a list, instead of having to know each service's name or IP address in advance. It is in everyone's interest for IP to have this capability. This is exactly the capability that Bonjour provides.

Zero Configuration: An Example

Zero-configuration IP networking holds a large amount of potential. Consider the everyday task of printing. Once a printer is configured on your computer or iOS device, it's simply a matter of choosing an application's Print command.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Take your laptop to a client's company, or a neighbor's house, and try to print something. If they have a printer that supports Bonjour protocols, printing is just as easy as it was on your local network. To print, connect your laptop to your client's Wi-Fi access point and start up your laptop. Or start up your laptop and it instantly finds your neighbor's home wireless network. Either way, your laptop automatically discovers any available printers. You open the document, choose the Print command, and every available printer appears in the Print dialog. You select a printer, click Print, and the document prints.

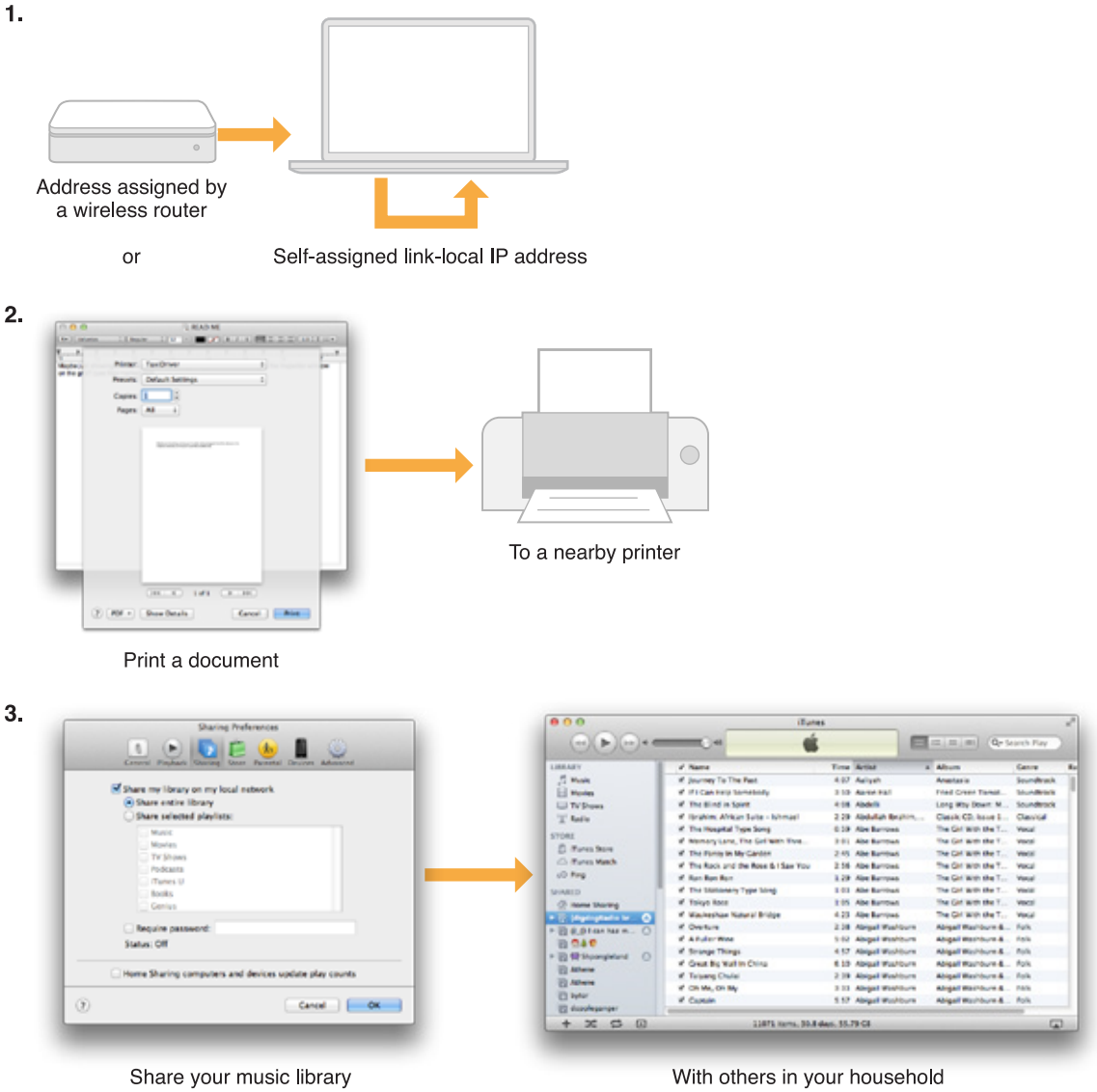
Or say you want to play a network game with a friend. You open the game, and your friend's copy of the game instantly sees your copy over the network. Or if you have a music sharing application on two computers, the programs themselves can discover each other and instantly swap song lists. Similarly, if you have a shared folder or have personal Web sharing turned on, your shared files and Web pages are instantly available to others.

This scenario is illustrated in Figure 1-1 (page 10). In step 1, you open up your laptop in your neighbor's house, and the laptop either obtains an address from the DHCP server in the router or, in the absence of a DHCP server, assigns itself an available local address. In step 2, the network is queried for available printers so that when you open the Print dialog, your neighbor's printer is listed. Finally, in step 3, you turn on music sharing on your computer, and your neighbor's computer sees it and connects.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

These are just a few of the existing applications that can benefit from zero-configuration IP networking. Zero-configuration IP networking has the potential to enhance mobile gaming, in-home networking, distributed computing, and many other network applications. Additionally, zero-configuration IP networking opens the door for a whole new class of IP-enabled digital devices.

Figure 1-1 A typical zero-configuration networking session



What Is Bonjour?

Bonjour is Apple's proposal for zero-configuration networking over IP. Bonjour comes out of the work of the ZEROCONF Working Group, part of the Internet Engineering Task Force (IETF). The ZEROCONF Working Group's requirements and proposed solutions for zero-configuration networking over IP essentially cover three areas:

- addressing (allocating IP addresses to hosts)
- naming (using names to refer to hosts instead of IP addresses)
- service discovery (finding services on the network automatically)

Bonjour has a zero-configuration solution for all three of these areas, as described in the following four sections.

Bonjour allows service providers, hardware manufacturers, and application programmers to support a single network protocol—IP—while breaking new ground in ease of use.

Network users no longer have to assign IP addresses, assign host names, or even type in names to access services on the network. Users simply ask to see what network services are available, and choose from the list.

In many ways, this kind of browsing is even more powerful for applications than for users. Applications can automatically detect services they need or other applications they can interact with, allowing automatic connection, communication, and data exchange, without requiring user intervention.

Addressing

The addressing problem is solved by self-assigned link-local addressing. Link-local addressing uses a range of addresses reserved for the local network, typically a small LAN or a single LAN segment. To that end, the IPv6 specification includes self-assigned link-local addressing as part of the protocol. The main addressing challenge in zero-configuration networking was retrofitting this capability to IPv4.

> **Note:** IPv6 link-local addressing is simpler than IPv4 link-local addressing, and thus is more reliable. Because of this, it is important that your app support IPv6.

In IPv4, self-assigned addressing works by picking a random IP address in the link-local range and testing it. If the address is not in use, it becomes your local address. If it is already in use, the computer or other device chooses another address at random and tries again.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

> **Note:** Two hosts are considered to be on the same local link if, when one host sends packets to the other, the entire link-layer payload (the content of the packet as represented in the physical network, such as Ethernet) arrives unmodified. In practice, on an Ethernet network, this means that no IP router touches the packet between the two hosts.

Link-local addressing in IPv4 and IPv6 is supported on most major operating systems. Hardware manufacturers should implement link-local addressing on their devices to obtain the full benefit of Bonjour.

Any user or service on a computer or iOS device that supports link-local addressing benefits from this feature automatically. When your host computer encounters a local network, it finds an unused local address and adopts it. No action on your part is required.

Naming

The proposed solution for name-to-address translation on a local network uses Multicast DNS (mDNS), in which DNS-format queries are sent over the local network using IP multicast. Because these DNS queries are sent to a multicast address, no single DNS server with global knowledge is required to answer the queries. Each service or device can provide its own DNS capability—when it sees a query for its own name, it provides a DNS response with its own address.

Bonjour goes a bit further. It includes a responder that handles mDNS queries for any network service on the host computer or iOS device. This relieves your application of the need to interpret and respond to mDNS messages. By registering your service, the Bonjour mDNSResponder daemon automatically advertises the availability of your service so that any queries for your name are directed to the correct IP address and port number automatically.

> **Note:** Registration is performed using one of the Bonjour APIs. This functionality is available only to services running on the host computer or iOS device. Services running on other devices, such as printers, need to implement a simple mDNSResponder daemon that handles queries for services provided by that device.

Bonjour also provides built-in support for the NAT port mapping protocol (NAT-PMP). If the upstream router supports this protocol, OS X and iOS apps can create and destroy port mappings to allow hosts on the other side of the firewall to connect to the provided services. (NAT port mappings are described further in "Firewalls and Network Address Translation" in *Networking Overview*.)

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

For name-to-address translation to work properly, a unique name on the local network is necessary. Unlike conventional DNS host names, the local name only has significance on the local network or LAN segment. You can self-assign a local name the same way you self-assign a local address—choose one; if it's not already in use, it's yours:

- Hardware manufacturers determine whether their chosen name is already in use by having their device send an mDNS query for that name and looking for any response. If there is a response, the device should choose another name. Devices without a user interface append an incrementally larger number to a default name until the name is unique. For example, if a printer with the default name `XYZ-LaserPrinter.local` attaches to a local network with two other identical printers already installed, it tests for `XYZ-LaserPrinter.local`, then `XYZ-LaserPrinter-2.local`, then `XYZ-LaserPrinter-3.local`, which is unused and becomes its name.
- Software services provide a name when they register with Bonjour. If the provided name is already in use, Bonjour will automatically rename your service for you by default.

In OS X, users can set a host name for their computers via the Local Hostname setting in the Sharing pane of System Preferences. (In iOS, the host name is generated automatically and is not configurable.) This host name can be used anywhere a conventional DNS host name is used—Web browsers, command line tools, and so on. To indicate to the system that a name is a local host name, append a dot (`.`) and `local.` to the host name; `Steve.local.` is an example of a local host name.

Important: The first dot acts as a separator. To prevent applications from looking for services using the search domain, fully enumerate a host name by adding the last dot in `local.`.

For example, if a user types `steve.local.` into a Web browser, this tells the system to multicast the request for `steve` on the local network instead of sending it to the conventional DNS server. If a Bonjour-enabled computer named `steve` is on the local network, the user's browser is sent the correct IP address for it. This allows users to access local hosts and services without a conventional DNS server.

Note: Users can avoid typing `.local.` after Bonjour host names by entering `local` in the Search Domains section of the Network pane in System Preferences, along with any other DNS domains such as `apple.com` or `earthlink.net`. An unqualified name, such as `steve`, is searched for in successive domains listed in the Search Domains section of the Network pane, in this case `steve.apple.com`, `steve.earthlink.net`, and `steve.local`.

For more information, see "Domain Naming Conventions" (page 17).

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Service Discovery

The final element of Bonjour is service discovery. Service discovery allows applications to find all available instances of a particular type of service and to maintain a list of named services and port numbers. The application can then resolve the service hostname to a list of IPv4 and IPv6 addresses, as described in "Naming" (page 12).

The list of named services provides a layer of indirection between a service and its current DNS name and port number. Indirection allows applications keep a persistent list of available services and resolve an actual network address just prior to using a service. The list allows services to be relocated dynamically without generating a lot of network traffic announcing the change.

Service discovery in Bonjour is accomplished by "browsing." An mDNS query is sent out for a given service type and domain, and any matching services reply with their names. The result is a list of available services to choose from.

This is very different from the traditional device-centric idea of network services. For someone who deals with servers, network devices, and network programming, it is easy to get in the habit of thinking about services in terms of physical hardware. In this device-centric view, the network consists of a number of devices or hosts, each with a set of services. For example, the network might consist of a server machine and several client machines. In a device-centric browsing scheme, a client queries the server for what services it is running, gets back a list (FTP, HTTP, and so on), and decides which service to use. The interface reflects the way the physical system is organized. But this is not necessarily what the user logically wants or needs.

Users typically want to accomplish a certain task, not query a list of devices to find out what services are running. It makes far more sense for a client to ask a single question: "What print services are available?" than to query each available device with the question, "What services are you running?" and sift through the results looking for printers. The device-centric approach is not only time-consuming, it generates a tremendous amount of network traffic, most of it useless. The service-centric approach sends a single query, generating only relevant replies.

Additionally, services are not tied to specific IP addresses or even host names. For example, a website may be hosted by multiple servers with different addresses. Within an organization, network administrators may need to move a service from one server to another to help balance the load. If clients store the host name (as in most cases they now do), they will not be able to connect if the service moves to a different host.

Bonjour takes the service-oriented view. Queries are made according to the type of service needed, not the hosts providing them. Applications store service instance names, not addresses, so if the IP address, port number, or even host name has changed, the application can still connect. By concentrating on services rather than devices, the user's browsing experience is made more useful and trouble-free.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

How Bonjour Reduces Overhead

Server-free addressing, naming, and service discovery have the potential to create a significant amount of excess network traffic, but Bonjour takes a number of steps to reduce this traffic to a minimum. This allows Bonjour to attain AppleTalk's ease of use while avoiding any unnecessary "chattiness."

Bonjour makes use of several mechanisms for reducing zero-configuration overhead, including caching, suppression of duplicate responses, exponential back-off, and service announcement, as described in the following sections.

Caching

Bonjour uses a cache of Multicast DNS records to prevent hosts from requesting information that has already been requested. For example, when one host requests, say, a list of LPR print spoolers, the list of printers comes back via multicast, so all local hosts see it. The next time a host needs a list of print spoolers, it already has the list in its cache and does not need to reissue the query. The Multicast DNS responder is responsible for maintaining the cache; application developers do not need to do anything to maintain it.

Suppression of Duplicate Responses

To prevent repeated answers to the same query, Bonjour service queries include a list of known answers. For example, if a host is browsing for printers, the first query includes no print services and gets, say, twelve replies from available print servers. The next time the host queries for print services, the query includes a list of known servers. Print servers already on the list do not respond.

Bonjour suppresses duplicate responses in another way. If a host is about to respond, and notices that another host has already responded with the same information, the host suppresses its response.

Application developers do not need to take any action to suppress duplicate responses. Bonjour handles duplicate response suppression.

Exponential Back-off and Service Announcement

When a host is browsing for services, it does not continually send queries to see if new services are available. Instead, the host issues an initial query and sends subsequent queries exponentially less often, for example: after 1 second, 3 seconds, 9 seconds, 27 seconds, and so on, up to a maximum interval of one hour.

This does not mean that it can take over an hour for a browser to see a new service. When a service starts up on the network, it announces its presence a few times using a similar exponential back-off algorithm. This way, network traffic for service announcement and discovery is kept to a minimum, but new services are seen very quickly.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Services running on a Bonjour-equipped host are announced automatically when they register with the mDNSResponder daemon. Services running on other hardware, such as printers, should implement service announcement with exponential back-off to take full advantage of Bonjour.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

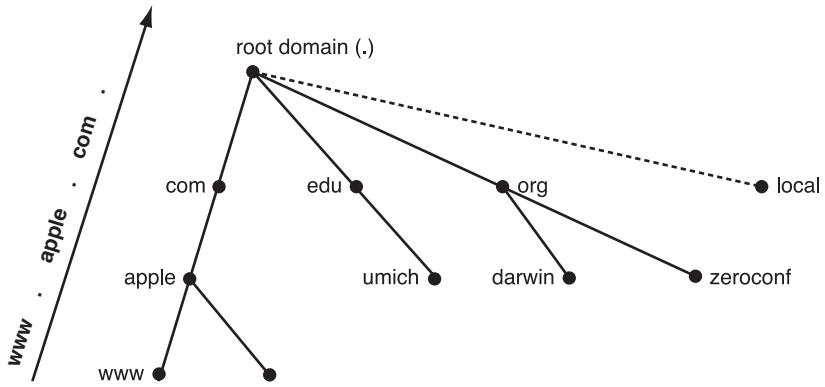
Domain Naming Conventions

Bonjour names for service instances and service types are related to Domain Name System (DNS) domain names. This section explains DNS domain names, the Bonjour local "domain," and the naming rules for Bonjour service instances and service types.

Domain Names and DNS

DNS uses a specific-to-general naming scheme for domain names. The most general domain is `.` ("dot"), called the **root domain**, which is akin to the root directory `/` in a UNIX file system. Every other domain falls in a hierarchy below the root domain. For example, the name `www.apple.com.` is within the **second-level domain** `apple.com.`, which is within the **top-level domain** `com.`, which in turn is part of `.` ("dot"), the root domain. Figure 2-1 shows an abridged version of this hierarchy.

Figure 2-1 Part of the Internet Domain Name System, augmented for Bonjour



At the top of the inverted tree is the root domain. Below it are some of the top-level domains: `com.`, `edu.`, and `org.`, and the local Bonjour "domain" `local.`, discussed further in "Bonjour and the Local Link" (page 18). Below the top level are a few second-level domains, `apple`, `darwin`, and `zeroconf`. The tree can extend infinitely downward with, for example, `www`, at the third level.

You may have noticed that the trailing dot is left off of most domain names. The trailing dot does, however, have meaning. A domain name ending in a trailing dot, such as `www.apple.com.`, is known as a **fully qualified domain name**, much like an absolute path (such as `/usr/bin`) in a UNIX file system.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

If you type `wibble.apple.com` into your Web browser (without the trailing dot), the system treats it as an unqualified (partial) name and appends the names from your list of search domains, such as `example.com.`, `example.edu`, and so on. The system first tries to append `.` ("dot," the root domain), but if the name `wibble.apple.com.` doesn't exist, it continues down the list and tries `wibble.apple.com.example.com.`, `wibble.apple.com.example.edu.`, and so on. Although this search domain feature is often useful, it is probably not what you intended in this case.

Bonjour and the Local Link

Bonjour protocols deal in large part with the part of the network called the **local link**. A host's local link, or **link-local network**, includes itself and all other hosts that can exchange packets without IP header data being modified. In practice, this includes all hosts not separated by a router.

On Bonjour systems, `local.` is used to indicate a name that should be looked up using an IP multicast query on the local IP network.

Note that `local.` is not really a domain. You can think of `local.` as a pseudo-domain. It differs from conventional DNS domains in a fundamental way: names within other domains are globally unique; link-local domain names are not. There is only one logical DNS entry in the world named `www.apple.com.`, and because of the way DNS works, there can be only one. Host names ending in `local.`, on the other hand, are managed by a collection of Multicast DNS responders on the local network, so the naming scope is just that: local. There can easily be two hosts named `meow.local.` in the world, or even the same building, just not on the same local network.

Globally unique names are important and useful—in fact, they are one of the significant achievements of the Internet—but they require a certain level of administrative effort to set up and maintain. Local names are useful only on the local network, but in cases where that is adequate, they provide a way to refer to network devices using names instead of IP numbers, and of course they require less effort and expense to coordinate than globally unique names.

Locally unique names are particularly useful on networks that have no connection to the global Internet, either by design or because of interruption, and on small, temporary networks, such as a pair of computers linked by a crossover cable, or a few people playing network games using laptops on the wireless network of a home or cafe.

If a name collision on the local network occurs, a Bonjour host finds a new name automatically (in the case of iOS or any device without a screen) or by asking the user (in the case of a personal computer).

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Bonjour and Unicast DNS

In addition to multicast DNS, Bonjour supports advertising and discovering services over traditional unicast DNS using wide-area Bonjour. Unicast DNS is outside the scope of this document. To learn more about configuring unicast domain name servers for use with Bonjour, see the DNS-SD website.

Bonjour Names for Existing Service Types

Bonjour services are named according to the existing Internet standard for IP services (described in RFC 2782). Bonjour service names combine service types and transport protocols to form a registration type. The registration type is used to register a service and create DNS resource records for it. To distinguish registration types from domain names in DNS resource records, registration types use underscore prefixes to separate the components that make up a registration type. The format is

_ServiceType `._` *_TransportProtocolName* `.`

The service type is the official IANA-registered name for the service, for example, `ftp`, `http` or `printer`. The transport protocol name is `tcp` or `udp`, depending on the transport protocol the service uses. An FTP service running over TCP would have a registration type of `_ftp._tcp`. and would register a DNS PTR record named `_ftp._tcp.local.` with its hosts' Multicast DNS responder.

Bonjour Names for New Services

If you are designing a new protocol to advertise as a Bonjour network service, you should register it with IANA.

The IANA currently requires that every registered service be associated with a "well-known port" or range of well-known ports. For example, `http` is assigned port `80`, so that whenever you visit a website in your web browser, the application assumes that the HTTP service is running on port `80` unless you specify otherwise. This way, the port number for a website need only be memorized if the website is configured in a non-standard way.

With Bonjour, however, you don't have to know about port numbers. Because client applications can discover your service with a simple query for the service type, well-known ports are unnecessary.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

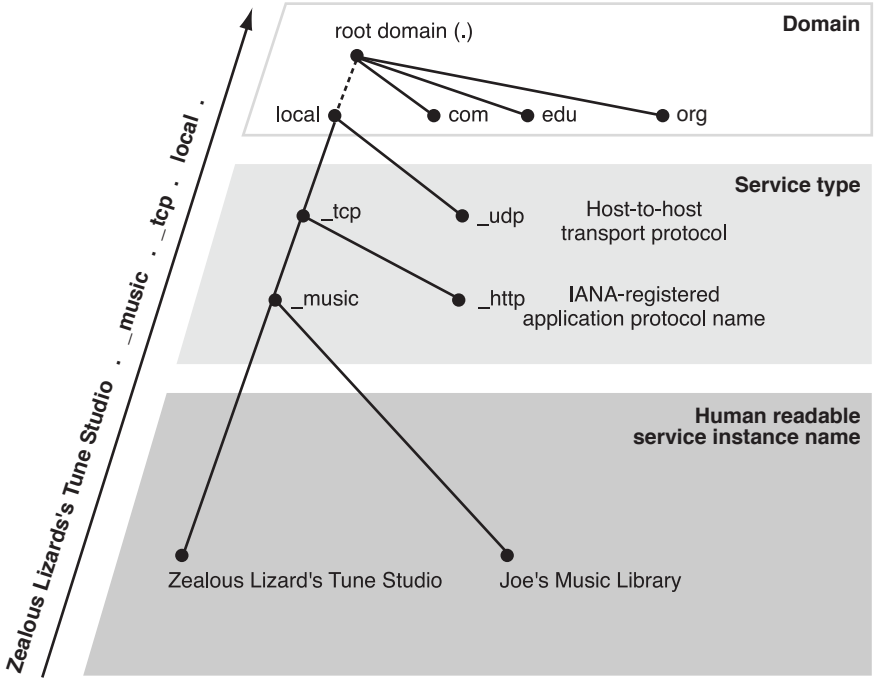
Bonjour Names for Service Instances

Service instance names are intended to be human-readable strings. As such, you should name them descriptively, and let the user override whatever default name you provide. Because they are intended to be browsed rather than typed, service instance names can be any Unicode string encoded with UTF-8, up to 63 octets (bytes) in length.

For example, an application for sharing music over the network might use the local user's name for a music sharing service, such as `Émille's Music Library`, by default. The user could override the default and name the service `Zealous Lizard's Tune Studio`, and the application would register a DNS SRV record named `Zealous Lizard's Tune Studio._music._tcp.local.`, assuming the application's music sharing protocol was associated with the name `music`.

Figure 2-2 (page 20) illustrates the organization of the name of a Bonjour service instance. At the top level of the tree is the domain, such as `local.` for the local network. Below the domain is the registration type, which consists of the service type preceded by an underscore (`_music`) and the transport protocol, also preceded by an underscore (`_tcp`). At the bottom of the tree is the human-readable service instance name, such as `Zealous Lizard's Tune Studio`. The complete name is a path along the tree from bottom to top, with each component separated by a dot.

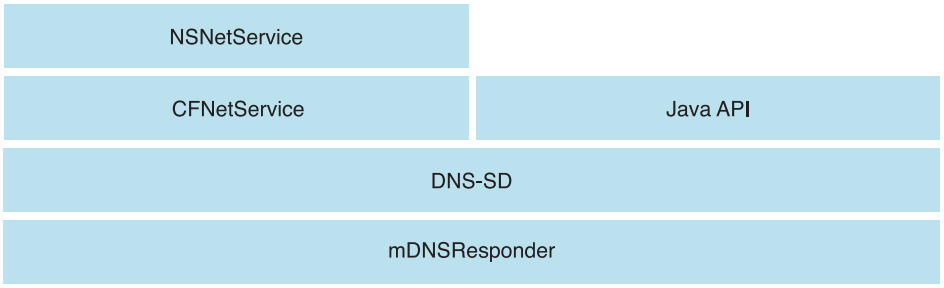
Figure 2-2 Organization of a Bonjour service name



Bonjour API Architecture

OS X and iOS provide several layers of application programming interface (API) for Bonjour service applications: The NSNetService and NSNetServiceBrowser classes in the Foundation framework; CFNetServices, part of the CFNetwork framework in Core Services; DNS Service Discovery for Java (OS X only); and the low-level DNS Service Discovery API built around BSD sockets. All three API sets provide facilities for publication, discovery, and resolution of network services. Figure 3-1 (page 21) illustrates the structure of the API layers. As you can see, the Multicast DNS responder (or other DNS server) sits at the lowest level, so your software does not have to interact directly with DNS.

Figure 3-1 API layers for Bonjour network services



NSNetService and NSNetServiceBrowser

The `NSNetService` and `NSNetServiceBrowser` classes, part of the Foundation framework in Cocoa, provide object-oriented abstractions for service discovery and publication. `NSNetService` objects represent instances of Bonjour services, either for publication or a service discovered by a client, and `NSNetServiceBrowser` represents a browser for a particular type of service. Most Cocoa programmers should find these classes sufficient to meet their needs. If you need more detailed control, you can use the DNS Service Discovery API from a Cocoa application.

`NSNetService` and `NSNetServiceBrowser` are scheduled on the default `NSRunLoop` object to perform publication, discovery, and resolution asynchronously. All results returned by `NSNetService` and `NSNetServiceBrowser` objects are handled by delegate objects. These objects must be associated with a run loop to function, but it need not be the default one.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

CFNetServices

The CFNetServices API declared in the Core Services framework provide Core Foundation-style types and functions for managing services and service discovery. CFNetServices defines three Core Foundation object types, CFNetService, CFNetServiceBrowser, and CFNetServiceMonitor. CFNetService is an abstract representation of a service instance, either for publication or for use. Associated functions provide support for publishing and resolving services. CFNetServiceBrowser represents a browser for a particular type of service in a particular domain. You should generally use this API only if you are writing code at the Core Foundation layer in OS X or iOS.

Both CFNetService and CFNetServiceBrowser objects are normally serviced in CFRunLoops. To retrieve results, applications implement callback functions to handle events, such as new services appearing or disappearing, instances being resolved, and errors occurring. Unlike NSNetService and NSNetServiceBrowser, CFNetServices types do not require a run loop and can run synchronously if this behavior is needed. However, it is bad practice to use the synchronous modes of these functions.

DNS Service Discovery

The DNS Service Discovery API, declared in `/usr/include/dns_sd.h`, provide low-level BSD socket communication for Bonjour services. DNS Service Discovery acts as an intermediate layer between your software and the Multicast DNS responder or DNS server. It manages the Multicast DNS responder for you, letting you write your program in terms of services and service browsers instead of DNS resource records.

Because the DNS Service Discovery API is part of the Darwin open source project, you should use it when writing cross-platform code (for platforms beyond just iOS and OS X) or when you need to use low-level features that are unavailable in higher-level APIs such as `NSNetService`.

DNS Service Discovery is also the API that should be used if developing Bonjour service applications for Windows, Linux, or FreeBSD.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Bonjour Operations

This chapter describes the Bonjour operations of service publication, browsing, and resolution that underlay the three network service API layers, and the API layers themselves. You should read this chapter if you want to write an application or tool that publishes or discovers network services.

Architectural Overview

The network services architecture in Bonjour includes an easy-to-use mechanism for publishing, discovering, and using IP-based services. Bonjour supports three fundamental operations, each of which is a necessary part of zero-configuration network services:

- Publication (advertising a service)
- Discovery (browsing for available services)
- Resolution (translating service instance names to addresses and port numbers for use)

These operations are discussed in detail in the following sections.

Publication

To publish a service, an application or device must register the service with a Multicast DNS responder, either through a high-level API or by communicating directly with the responder (`mDNSResponder`). Bonjour also supports storing records on conventional DNS servers as well, using Dynamic DNS Update.

When a service is registered, three related DNS records are created: a service (SRV) record, a pointer (PTR) record, and a text (TXT) record. The TXT record contains additional data needed to resolve or use the service, although it is also often empty.

Service Records

The SRV record maps the name of the service instance to the information needed by a client to actually use the service. Clients then store the service instance name as a persistent way to access the service, and perform a DNS query for the host name and port number when it's time to connect. This additional level of indirection

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

provides for two important features. First, the service is identified by a human-readable name instead of a domain name and port number. Second, clients can access the service even if its port number, IP address, or host name changes, as long as the service name remains the same.

The SRV record contains two pieces of information to identify a service:

- Host name
- Port number

The host name is the domain name where the service can currently be found. The reason a host name is given instead of a single IP address is that it could be a multi-homed host with more than one IP address, or it could have IPv6 addresses as well as IPv4 addresses, and so on. Identifying the host by name allows all these cases to be handled gracefully.

The port number identifies the UDP or TCP port for the service.

SRV records are named according to the following convention:

```
<Instance Name>.<Service Type>.<Domain>
```

`<Instance Name>`, the name of a service instance, can be any UTF-8-encoded Unicode string, and is intended to be human readable.

`<Service Type>` is a standard IP protocol name, preceded by an underscore, followed by the host-to-host transport protocol (TCP or UDP), also preceded by an underscore. For example, a Trivial FTP service running over UDP would have a service type of `_tftp._udp`, and an IPP printing service running over TCP would register under the `_ipp._tcp` service type. The list of official protocol names is maintained by IANA; see "Domain Naming Conventions" (page 17) for more information.

`<Domain>` is a standard DNS domain. This may be a specific domain, such as `apple.com.`, or the generic suffix `local.` for a service accessible only on the local link.

Here is an example of the SRV record (in the standard DNS record format) for a print spooler named `PrintsAlot` running on TCP port `515`:

```
PrintsAlot._printer._tcp.local. 120 IN SRV 0 0 515 blackhawk.local.
```

This record would be created on the Multicast DNS responder of a printer called `blackhawk.local.` on the local link. The initial `120` represents the time-to-live (TTL) value which is used for caching. The two zeros are weight and priority values, used in traditional DNS when choosing between multiple records that match a given name; for multicast DNS purposes, these values are ignored.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

For more information about domain, service, and instance names, see "Domain Naming Conventions" (page 17).

Pointer Records

PTR records enable service discovery by mapping the type of the service to a list of names of specific instances of that type of service. This record adds yet another layer of indirection so services can be found just by looking up PTR records labeled with the service type.

The record contains just one piece of information, the name of the service instance (which is the same as the name of the SRV record). PTR records are accordingly named just like SRV records but without the instance name:

```
<Service Type>.<Domain>
```

Here is an example of a PTR record for a print spooler named `PrintsAlot`:

```
_printer._tcp.local. 28800 PTR PrintsAlot._printer._tcp.local.
```

Again, the `28800` is the time-to-live (TTL) value, measured in seconds.

Text Records

The TXT record has the same name as the corresponding SRV record, and can contain a small amount of additional information about the service instance, typically no more than 100–200 bytes at most. This record may also be empty. For example, a network game could advertise the name of the map being used in a multiplayer game, and a chat program could advertise the availability of the user (for example, idle, away, or available). If you need to transmit larger amounts of data, the host should establish a connection with the client and send the data directly.

Historically, this record has been used for multiple services running on the same port at the same IP address, for example multiple print queues running on the same print server. In this case additional information in the TXT record can be used to identify the intended print queue, as shown in this example:

Table 4-1 Using TXT records for demultiplexing

Service name	Type	IP address	Port	Queue name (from TXT record)
`Paper Printer`	`_ipp._tcp`	`10.0.0.2`	`631`	`rp=lpt1`
`Slide Printer`	`_ipp._tcp`	`10.0.0.2`	`631`	`rp=lpt2`

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

This kind of practice was necessary because service types have historically been associated with well known ports. Designers of new Bonjour protocols are encouraged to run each instance of their service on a different dynamically allocated port number, instead of trying to run them all on the same well known port number and using extra information to specify which instance the client is trying to talk to.

The nature and format of the data in the TXT record are specific to each type of service, so each new service type needs to also define the format of data for its associated TXT record (if any), and publish this format as part of the protocol specification.

Publication: An Example

For a concrete example, consider a hypothetical device that shares music over a local network—an IP-enabled jukebox. Suppose that its transport protocol is TCP and its application protocol goes by the name `music`. When someone plugs the device into an Ethernet hub, a number of things happen, as shown in Figure 4-1.

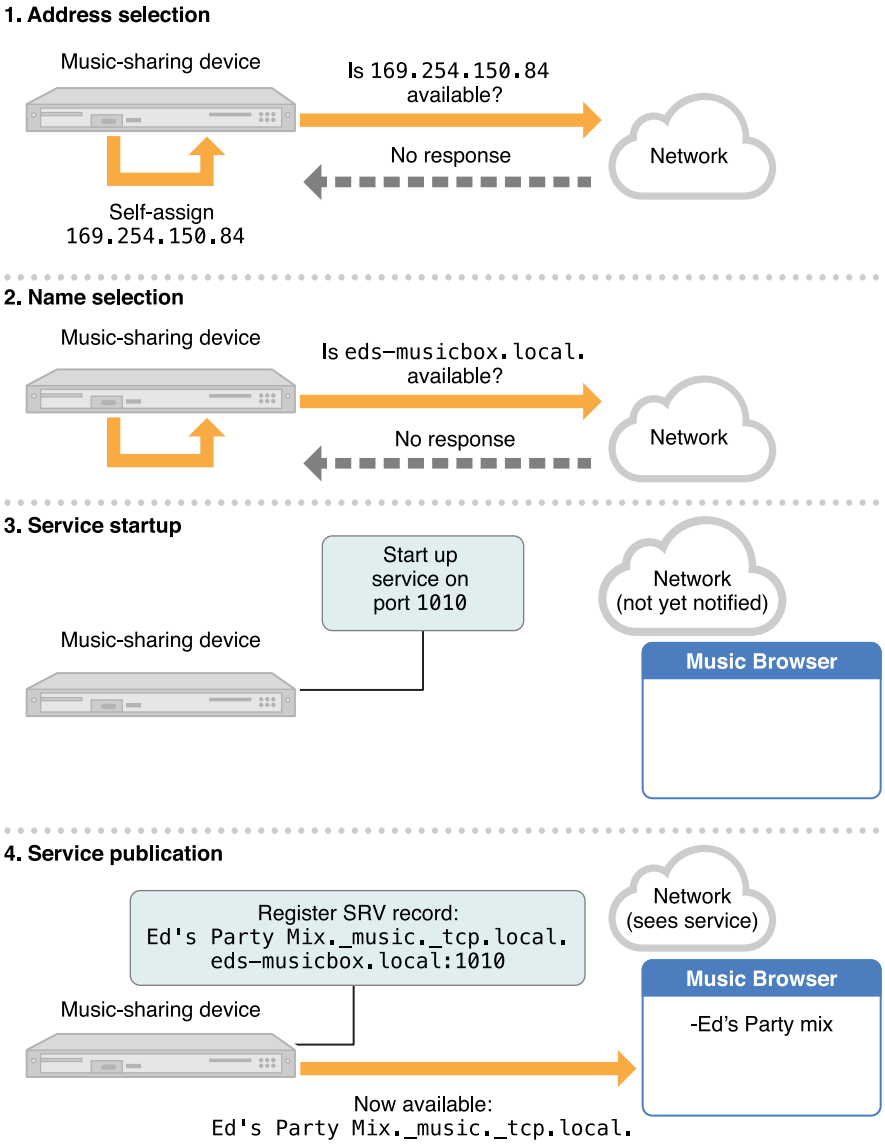
In step 1, the device randomly selects the link-local IP address `169.254.150.84`, randomly selected from the IPv4 link-local range `169.254.0.0` with a subnet mask of `255.255.0.0`, and announces it to the network. Because no devices respond to the announcement, the device takes the address as its own. In step 2, it starts up its own Multicast DNS responder, requests the host name `eds-musicbox.local.`, verifies its availability, and takes the name as its own. Next, in step 3, the device starts up a music sharing service on TCP port `1010`. Finally, in step 4, it publishes the service, of type `_music._tcp`, under the name `Ed's Party Mix`, in the `local.` domain, first making sure that no service exists under the same name. This creates two records:

- An SRV record named `Ed's Party Mix._music._tcp.local.` that points to `eds-musicbox.local.` on TCP port `1010`

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

- A PTR record named `_music._tcp.local.` that points to the `Ed's Party Mix._music._tcp.local.` service.

Figure 4-1 Publishing a music sharing service



Note: When publishing a service, if the domain name or service name is unavailable, a device without an interface should choose a new name. Application software encountering this situation should present a user interface informing the user that the name is unavailable, and should allow the user to choose a different name.

Discovery

Service discovery makes use of the DNS records registered during service publication to find all named instances of a particular type of service. To do this, an application performs a query for PTR records matching a service type, such as `_http._tcp`, usually through a higher-level API. The Multicast DNS responders running on each device return PTR records with service instance names.

Figure 4-2 Discovering music sharing services

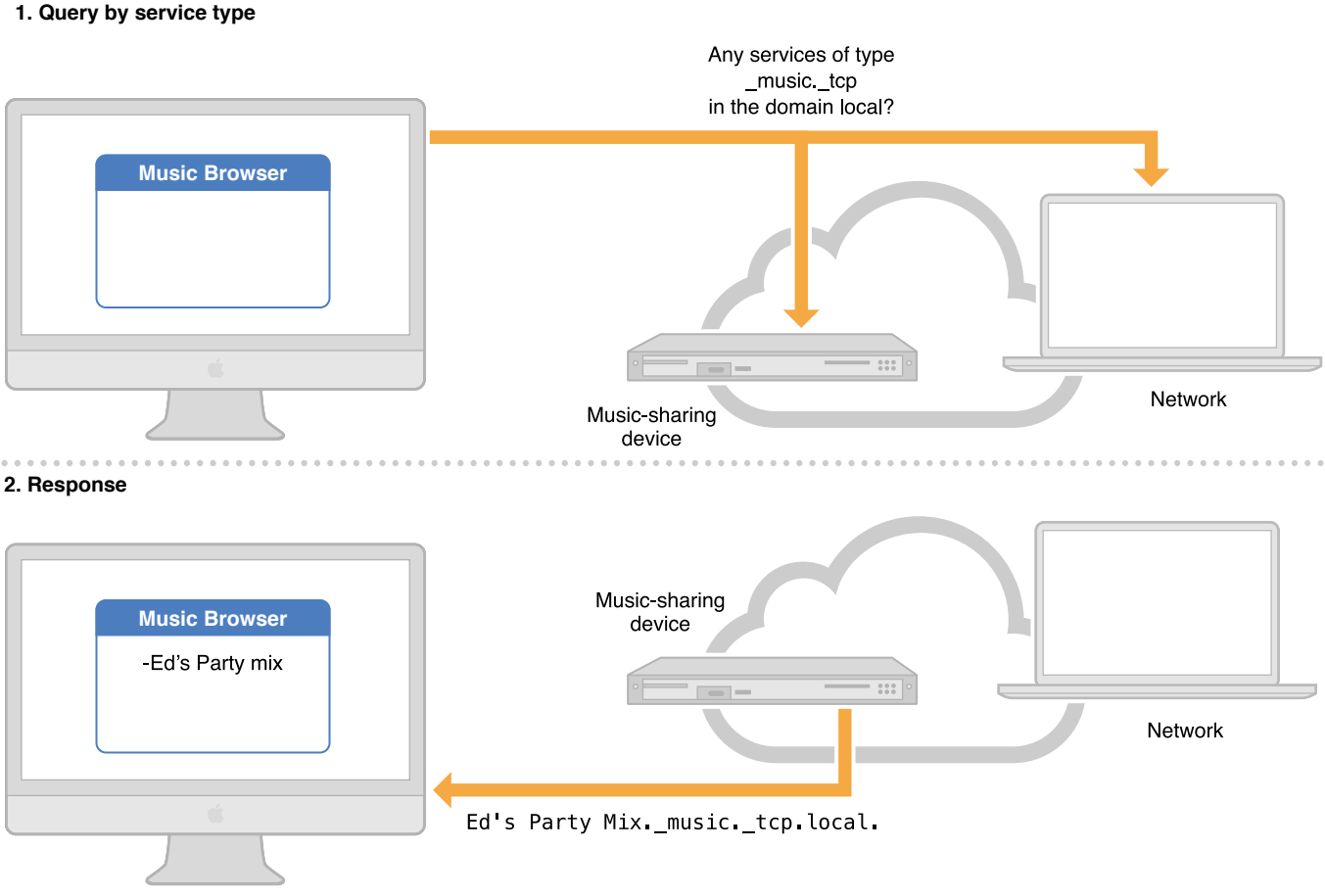


Figure 4-2 illustrates a client application browsing for music sharing services. In step 1, the client application issues a query for services of type `_music._tcp` in the `local.` domain to the standard multicast address `224.0.0.251`. Every Multicast DNS responder on the network hears the request, but only the music sharing device responds with a PTR record (step 2). The resulting PTR record holds the service instance name, `Ed's Party Mix._music._tcp.local.` in this case. The client app can then extract the service instance name from the PTR record and add it to an onscreen list of music servers.

Resolution

Service discovery typically takes place only once in a while—for example, when a user first selects a printer. This operation saves the service instance name, the intended stable identifier for any given instance of a service. Port numbers, IP addresses, and even host names can change from day to day, but a user should not need to reselect a printer every time this happens. Accordingly, resolution from a service name to socket information does not happen until the service is actually used.

To resolve a service, an application performs a DNS lookup for a SRV record with the name of the service. The Multicast DNS responder responds with the SRV record containing the current information.

Figure 4-3 illustrates service resolution in the music sharing example. The resolution process begins with a DNS query to the multicast address `224.0.0.251` asking for the `Ed’s Party Mix._music._tcp.local.` SRV record (step 1). In step 2, this query returns the service’s host name and port number (`eds–musicbox.local.`, `1010`). In step 3, the client sends out a multicast request for the IP address. In step

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

4, this request resolves to the IP address `169.254.150.84`. Then the client can use the IP address and port number to connect to the service. This process takes place each time the service is used, thereby always finding the service's most current address and port number.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Figure 4-3 Resolving a music sharing service instance

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Bonjour - Frequently Asked Questions

This appendix describes frequently asked questions about Bonjour.

1. What is Bonjour?

Bonjour, also known as zero-configuration networking, enables automatic discovery of computers, devices, and services on IP networks. Bonjour uses industry standard IP protocols to allow devices to automatically discover each other without the need to enter IP addresses or configure DNS servers. Specifically, Bonjour enables automatic IP address assignment without a DHCP server, name to address translation without a DNS server, and service discovery without a directory server. Bonjour is an open protocol which Apple has submitted to the IETF as part of the ongoing standards-creation process. To learn more, check out the Bonjour Protocol Specifications which detail the technologies that make up Link-Local and Wide-Area Bonjour.

2. What is mDNSResponder?

mDNSResponder is a Bonjour system service that implements Multicast DNS Service Discovery for discovery of services on the local network, and Unicast DNS Service Discovery for discovery of services anywhere in the world. mDNSResponder is built into OS X and iOS and can be downloaded as part of Bonjour for Windows. Applications like iTunes, iPhoto, Messages and Safari use mDNSResponder to implement zero-configuration network music sharing, photo sharing, chatting and file sharing, and discovery of remote user interfaces for hardware devices like printers and web cameras. mDNSResponder is also used to discover and print to Bonjour printers and USB printers connected to the AirPort Extreme and Express base stations. mDNSResponder is open source, and hardware device manufacturers are encouraged to embed the mDNSResponder source code directly into their products to benefit from zero-configuration networking.

3. Does Bonjour work between multiple subnets?

Yes. The first release of DNS Service Discovery (DNS-SD) for OS X concentrated on Multicast DNS (mDNS) for single-link networks because this was the environment worst served by IP software. Bonjour uses Dynamic DNS Update (RFC 2316) and unicast DNS queries to enable wide-area service discovery.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

4. When I disconnect a device from a network, does it remain visible?

Yes, for a while. Eventually, the DNS record reaches its time-to-live interval and disappears. As an app developer, if you connect to a host using Bonjour and the connection fails, you can ask the Bonjour to reconfirm the record. This process is described further in *NSNetServices and CFNetServices Programming Guide*.

5. What do I have to do to support Bonjour over Bluetooth in iOS?

In iOS 5 and later, apps must explicitly opt in to doing service discovery over Bluetooth, and must resolve services using the low-level DNS Service Discovery C API. For more information, see *Bonjour over Bluetooth on iOS 5.0*.

6. How long should I leave service browsers running?

Browsers consume resources, so you should not keep them running if you don't expect to ever use the data. However, keeping a service browser running while connecting to a service is generally a good idea. If that connection fails, the presence of a running browser encourages Bonjour to more aggressively revalidate potentially stale service entries, which can make the list of services more accurate.

As a rule, if you are not showing any user interface elements that contain the list, and if you are not actively connecting to any service, you should probably stop the browser. However, this is just a general recommendation; in all cases, you should do whatever results in the best experience for your users.

7. Does Bonjour support "SOAP" RPC over HTTP?

Yes. Bonjour defines a new protocol for discovering services (DNS-SD), however, it places no restrictions on the type of services you discover. Thus you can discover SOAP services just as easily as you can discover Messages buddies and iTunes music libraries. In other words, Bonjour supports SOAP over HTTP as well as every other application protocol layered on top of TCP/IP or UDP/IP.

8. Does Bonjour have any kind of subscription or notification mechanism?

Yes. The reason that many people seem to be unaware that Bonjour also does notification is probably because it is simply an intrinsic property of the discovery protocol. With a well-designed discovery protocol, the same protocol that you use to discover some piece of information is also used to discover changes to that information.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Discovery of static information, and discovery of variable information, and discovering when variable information changes are all just different points on the same spectrum. For an example of an application using Bonjour "notifications", check out Messages. When you change your Status from "Available" to "Away" or type in a status message, all other Messages clients on the local network are notified of the change.

This technique is described further in *NSNetServices and CFNetServices Programming Guide*.

9. What should I pass in for the "name" parameter when registering a service?

By default, you should choose a human-readable name that uniquely describes the service. iTunes, for example, chooses a default music sharing name by combining the computer user's first and last name, as in "Isaac Newton's Music". For most hardware devices, the default service name should be the full make and model of the product. For example, something like "Apple MacBook Pro". Remember, this is only the default name given out of the box, and the user should be allowed to customize the service name to differentiate multiple devices or services on the network.

For OS X application developers that are registering services, it may make sense to have one instance of that service on a given computer (as opposed to one per instance of an application that may be running in multiple accounts). In that case, rather than having your application present its own user-interface for the user to enter the name of the advertised service, it's more convenient to register using the system-provided default name known as the "Computer Name" in the Sharing Preference Panel. If you pass in an empty string ("") for the service name when registering, the system will automatically use the "Computer Name". Passing in an empty string will also handle name conflicts by automatically appending a digit to the end of the name.

However, there are services where multiple instances may be hosted on the same computer. For example, a print server with three printers should advertise each printer as a first-class entity. Each printer should be advertised using a descriptive name that usefully identifies the printer itself. This is important, because the printer called "Marketing's Transparency Printer" might get moved to a different print server in the future, but the user shouldn't have to be aware of those operational details. They should still see the same service advertised on the network under the same name, even though it's on a different print server now.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

10. What should I pass in for the "type" parameter when registering a service?

You must pass a string of the form `"_applicationprotocol._transportprotocol"`. Currently `"_transportprotocol"` must be either `"_tcp"` or `"_udp"`. Your `"applicationprotocol"` must be **15 characters or less** and should be registered with with IANA so they can add you to the list of registered protocol names and port numbers. Please see QA1312 for the list of service types used by OS X.

11. What should I pass in for the "domain" parameter when registering a service?

If you pass an empty string (`""`), then your service is registered using link-local multicast and in a user-chosen unicast DNS domain as well, if applicable.

If you pass in `"local"`, then your service is registered only using link-local multicast, and not in any user-chosen unicast DNS domains.

Except for the `"local"` domain, you only need to pass a specific string if you have some particular reason to want to register your service in a specific remote domain.

12. What should happen when two devices on the network both use the same service name?

In the rare case where a name collision occurs, your device should add a digit to the end of the name, for example:

"Apple Mac mini (2)"

Applications and devices that call Bonjour APIs like DNSServiceRegister and CFNetServiceRegisterWithOptions will automatically get this name changing behavior when name conflicts occur. For devices that have a screen and are capable of user input, instead of appending a number, you could optionally decide to prompt the user for a more unique name.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

13. What is the TXT record used for?

The specific nature of the TXT record and how it is to be used is service type dependent. Each service type will define zero or more name/value pairs used to store meta-data about each service. These name/value pairs should be formatted as described in Section 6 of DNS-Based Service Discovery. Please see QA1306 for information on how TXT records should be formatted when using the OS X and iOS APIs. Also, the DNS-SD Web Site has information regarding the currently defined name/value pairs for common service types.

14. After the user browses the network in my application and selects the service instance they wish to use, I should save that IP address in my application's preference file, right?

Wrong. This is a common mistake. With DHCP (and with link-local addressing) it is not safe to assume that the service instance will have the same IP address tomorrow. Addresses can change. Service names are the intended stable identifiers for a service instance. Save the instance name (name, type, and domain) in your application's preference file, then resolve it on demand each time the user accesses the service. Note also that you should not store the host name and port number, because you shouldn't assume that the service instance will necessarily be running on the same port number tomorrow. Instead of storing the host name, store the service instance name (name, type, and domain) and then when you resolve the service instance name at the time of use you are sure to get the up-to-date IP address and port number.

15. My hardware device has a built-in web server used for configuration. Should I register it using Bonjour?

Yes. You should register every service running on your device, for example, HTTP, FTP, SSH, Telnet. On OS X, the Safari web browser can discover web servers advertised with Bonjour, and Internet Explorer on Windows can discover web servers when Bonjour for Windows is installed. Also, the Terminal application in OS X can discover FTP, SSH, and Telnet servers.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Bonjour Overview*.

Date	Notes
2013-04-23	Revised introduction.
2012-12-13	Updated artwork and corrected minor technical errors.
2011-03-15	Incorporated Bonjour FAQ.
2006-05-23	Updated Related Documents section.
2005-11-09	Removed Preliminary stamp.
2005-10-04	Changed to be an overview of the Bonjour technology.
2005-04-29	Changed "Rendezvous" to "Bonjour."
2004-08-31	Clarified use of empty string when resolving services and added details on how to correctly process the address-resolution delegate method of the NSNetService class. Corrected example in *Resolving and Using Network Services* to not use empty string to refer to the local domain. Added details on how to correctly process the `netServiceDidResolveAddress:` delegate method of NSNetService.
2004-03-09	Clarified in "Bonjour Operations" (page 23) that a TXT record is always created, but it may be empty.
2004-03-08	Reorganized topic introduction. Added pointer to *DNS Service Discovery API*.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Date	Notes
2002-11-12	Revision history was added to existing document.

2013-04-23 | Copyright © 2013 Apple Inc. All Rights Reserved.

Apple Inc.
Copyright © 2013 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AirPort, AirPort Extreme, AppleTalk, Bonjour, Cocoa, iPhoto, iTunes, Mac, MacBook, OS X, and Safari are trademarks of Apple Inc., registered in the U.S. and other countries.

Newton is a trademark of Apple Inc.

Java is a registered trademark of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.